Iterative Linear Solvers for Sparse Matrices

Kengo Nakajima
Information Technology Center, The University of Tokyo, Japan

2013 International Summer School on HPC Challenges in Computational Sciences
New York University, New York, NY
June 24-28, 2013
• Sparse Matrices
• Iterative Linear Solvers
  – Preconditioning
  – Parallel Iterative Linear Solvers
  – Multigrid Method
  – Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
There are a lot of topics and issues all of which I cannot cover. I just try to talk about my experiences in the area of scientific applications and parallel numerical algorithms, with some general introductions.
Finite-Element Method (FEM)

- One of the most popular numerical methods for solving PDE’s.
  - elements (meshes) & nodes (vertices)
- Consider the following 2D heat transfer problem:
  \[ \lambda \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + Q = 0 \]
  - 16 nodes, 9 bi-linear elements
  - uniform thermal conductivity (\( \lambda = 1 \))
  - uniform volume heat flux (\( Q = 1 \))
  - \( T = 0 \) at node 1
  - Insulated boundaries
Galerkin FEM procedures

- Apply Galerkin procedures to each element:
  \[ \int_V [N]^T \left\{ \lambda \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + Q \right\} dV = 0 \]
  \[ \text{where } T = [N]\{\phi\} \text{ in each elem.} \]
  \[ \{\phi\} : T \text{ at each vertex} \]
  \[ [N] : \text{Shape function (Interpolation function)} \]

- Introduce the following “weak form” of original PDE using Green’s theorem:
  \[ - \int_V \left\{ \lambda \left( \frac{\partial [N]^T}{\partial x} \frac{\partial [N]}{\partial x} + \frac{\partial [N]^T}{\partial y} \frac{\partial [N]}{\partial y} \right) \right\} dV \cdot \{\phi\} \]
  \[ + \int_V Q[N]^T dV = 0 \]
Element Matrix

- Apply the integration to each element and form “element” matrix.

\[
- \int_{V} \lambda \left( \frac{\partial [N]^T}{\partial x} \frac{\partial [N]}{\partial x} + \frac{\partial [N]^T}{\partial y} \frac{\partial [N]}{\partial y} \right) dV \cdot \{\phi\} \\
+ \int_{V} Q[N]^T dV \, = \, 0
\]

\[
[k^{(e)}] \{\phi^{(e)}\} = \{f^{(e)}\}
\]

\[
\begin{bmatrix}
k_{AA}^{(e)} & k_{AB}^{(e)} & k_{AC}^{(e)} & k_{AD}^{(e)} \\
n_{BA}^{(e)} & k_{BB}^{(e)} & k_{BC}^{(e)} & k_{BD}^{(e)} \\
n_{CA}^{(e)} & n_{CB}^{(e)} & k_{CC}^{(e)} & k_{CD}^{(e)} \\
n_{DA}^{(e)} & n_{DB}^{(e)} & n_{DC}^{(e)} & k_{DD}^{(e)}
\end{bmatrix}
\begin{bmatrix}
\phi_{A}^{(e)} \\
\phi_{B}^{(e)} \\
\phi_{C}^{(e)} \\
\phi_{D}^{(e)}
\end{bmatrix}
= 
\begin{bmatrix}
f_{A}^{(e)} \\
f_{B}^{(e)} \\
f_{C}^{(e)} \\
f_{D}^{(e)}
\end{bmatrix}
\]
Global (Overall) Matrix
Accumulate each element matrix to “global” matrix.

\[
[K] \{\Phi\} = \{F\}
\]

\[
\begin{align*}
D & X & X & X \\
X & D & X & X & X \\
X & D & X & X & X \\
X & D & X & X & \\
X & X & X & D & X \\
X & X & X & X & X \\
X & X & X & X & X \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
X & X & X & X & \\
\end{align*}
\]

\[
\begin{pmatrix}
To each node ...
Effect of surrounding elem’s/nodes are accumulated.

\[
[K]\{\Phi\} = \{F\}
\]

\[
\begin{bmatrix}
D & X & X & X \\
X & D & X & X & X \\
X & D & X & X & X \\
X & D & X & X & X \\
X & X & D & X & X \\
X & X & D & X & X \\
X & X & D & X & X \\
X & X & D & X & X \\
X & X & D & X & X \\
X & X & D & X & X
\end{bmatrix}
\]

\[
\begin{aligned}
\Phi_1 & = F_1 \\
\Phi_2 & = F_2 \\
\Phi_3 & = F_3 \\
\Phi_4 & = F_4 \\
\Phi_5 & = F_5 \\
\Phi_6 & = F_6 \\
\Phi_7 & = F_7 \\
\Phi_8 & = F_8 \\
\Phi_9 & = F_9 \\
\Phi_{10} & = F_{10} \\
\Phi_{11} & = F_{11} \\
\Phi_{12} & = F_{12} \\
\Phi_{13} & = F_{13} \\
\Phi_{14} & = F_{14} \\
\Phi_{15} & = F_{15} \\
\Phi_{16} & = F_{16}
\end{aligned}
\]
Solve the obtained global eqn’s under certain boundary conditions ($\Phi_1 = 0$ in this case)

\[
\begin{bmatrix}
    D & X & X & X \\
    X & D & X & X & X \\
    X & D & X & X & X \\
    X & D & X & X \\
    X & X & D & X & X \\
    X & X & X & D & X & X \\
    X & X & X & D & X & X \\
    X & X & X & D & X & X \\
    X & X & X & D & X \\
    X & X & X & D & X \\
\end{bmatrix}
\begin{bmatrix}
    \Phi_1 \\
    \Phi_2 \\
    \Phi_3 \\
    \Phi_4 \\
    \Phi_5 \\
    \Phi_6 \\
    \Phi_7 \\
    \Phi_8 \\
    \Phi_9 \\
    \Phi_{10} \\
\end{bmatrix}
= \begin{bmatrix}
    F_1 \\
    F_2 \\
    F_3 \\
    F_4 \\
    F_5 \\
    F_6 \\
    F_7 \\
    F_8 \\
    F_9 \\
    F_{10} \\
\end{bmatrix}
\]
Result ...
2D FDM Mesh (5-point stencil)
Coef. Matrix derived from FDM/FEM

- **Sparse Matrix**
  - Many “0”s
- **Storing all components (e.g. \( A(i,j) \)) is not efficient for sparse matrices**
  - \( A(i,j) \) is suitable for dense matrices
- **Number of non-zero off-diagonal components is \( O(10^2) \) in FEM**
  - If number of unknowns is \( 10^8 \):
    - \( A(i,j) \): \( O(10^{16}) \) words
    - Actual Non-zero Components: \( O(10^{10}) \) words
- **Only (really) non-zero off-diag. components should be stored on memory**
Mat-Vec. Multiplication for Sparse Matrix Memory-Bound Process
Compressed Row Storage (CRS)

**Diag (i)** Diagonal Components (REAL, i=1~N)
**Index(i)** Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
**Item(k)** Off-Diagonal Components (Corresponding Column ID)
    (INT, k=1, index(N))
**AMat(k)** Off-Diagonal Components (Value)
    (REAL, k=1, index(N))

\[ \{Y\} = [A]\{X\} \]

```
do i = 1, N
    Y(i) = Diag(i) * X(i)
    do k = Index(i-1)+1, Index(i)
        Y(i) = Y(i) + AMat(k) * X(Item(k))
    enddo
enddo```

{Y} = [A] {X}
CRS or CSR?
for Compressed Row Storage

• In Japan and USA, “CRS” is very general for abbreviation of “Compressed Row Storage”, but they usually use “CSR” in Europe (especially in France).

• “CRS” in France
  – Compagnie Républicaine de Sécurité
    • Republic Security Company of France

• French scientists may feel uncomfortable when we use “CRS” in technical papers and/or presentations.
Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

\[
\begin{bmatrix}
  a_{11} & a_{12} & \ldots & a_{1,N-1} & a_{1,N} \\
  a_{21} & a_{22} & \ldots & a_{2,N-1} & a_{2,N} \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  a_{N-1,1} & a_{N-1,2} & \ldots & a_{N-1,N-1} & a_{N-1,N} \\
  a_{N,1} & a_{N,2} & \ldots & a_{N,N-1} & a_{N,N}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{N-1} \\
x_N
\end{bmatrix}
=
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_{N-1} \\
y_N
\end{bmatrix}
\]

\[
\{Y\} = [A] \{X\}
\]

\[
\text{do } j = 1, N \\
\text{Y}(j) = 0, \text{do0} \\
\text{do } i = 1, N \\
\text{Y}(j) = Y(j) + A(i, j) \times X(i) \\
\text{enddo} \\
\text{enddo}
\]
Compressed Row Storage (CRS): C
Numbering starts from 0 in program

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>○</td>
<td>1.1</td>
<td>2.4</td>
<td></td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4.3</td>
<td>3.6</td>
<td>2.5</td>
<td>3.7</td>
<td>9.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>5.7</td>
<td>1.5</td>
<td>3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4.1</td>
<td>9.8</td>
<td>2.5</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.1</td>
<td>9.5</td>
<td>10.4</td>
<td>11.5</td>
<td>4.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6.5</td>
<td>12.4</td>
<td>9.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6.4</td>
<td>2.5</td>
<td>1.4</td>
<td>23.1</td>
<td>13.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>9.5</td>
<td>1.3</td>
<td>9.6</td>
<td>3.1</td>
<td>51.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagonal Components
Diag[0] = 1.1
Diag[1] = 3.6
Diag[2] = 5.7
Diag[3] = 9.8
Diag[4] = 11.5
Diag[5] = 12.4
Diag[6] = 23.1
Diag[7] = 51.3

N = 8
## Compressed Row Storage (CRS): C

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.1</td>
<td>2.4</td>
<td></td>
<td></td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.6</td>
<td>4.3</td>
<td>2.5</td>
<td>3.7</td>
<td></td>
<td>9.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5.7</td>
<td></td>
<td></td>
<td>1.5</td>
<td>3.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9.8</td>
<td>4.1</td>
<td>2.5</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11.5</td>
<td>3.1</td>
<td>9.5</td>
<td>10.4</td>
<td>2.5</td>
<td></td>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12.4</td>
<td></td>
<td>6.5</td>
<td></td>
<td>1.4</td>
<td>9.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>23.1</td>
<td>6.4</td>
<td>2.5</td>
<td></td>
<td>1.3</td>
<td></td>
<td>13.1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>51.3</td>
<td>9.5</td>
<td>1.3</td>
<td>9.6</td>
<td>3.1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ISS-2013
### Compressed Row Storage (CRS): C

<table>
<thead>
<tr>
<th>Diag (i)</th>
<th>Index(i)</th>
<th>Item (k)</th>
<th>AMat(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal Components (REAL, i=1~N)</td>
<td>Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)</td>
<td>(Corresponding Column ID)</td>
<td>(Value)</td>
</tr>
</tbody>
</table>

\[
\{Y\} = [A]\{X\}
\]

for (i=0; i<N; i++)
\[
Y[i] = \text{Diag}[i]\times X[i];
\]
for (k=Index[i]; k<Index[i+1]; k++)
\[
Y[i] += \text{AMat}[k]\times X[\text{Item}[k]];
\]

#### Compressed Row Storage (CRS): C

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>3.6</td>
<td>9.8</td>
<td>11.5</td>
<td>12.4</td>
<td>23.1</td>
<td>51.3</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>1.0</td>
<td>1.8</td>
<td>3.1</td>
<td>4.1</td>
<td>6.4</td>
<td>9.5</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>4.3</td>
<td>4.9</td>
<td>6.7</td>
<td>9.5</td>
<td>6.5</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>2.5</td>
<td>5.4</td>
<td>9.1</td>
<td>10.4</td>
<td>1.2</td>
<td>11.1</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>3.3</td>
<td>5.1</td>
<td>2.7</td>
<td>4.3</td>
<td>5.9</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.7</td>
<td>4.1</td>
<td>2.5</td>
<td>4.1</td>
<td>9.8</td>
<td>6.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1</td>
<td>2.5</td>
<td>4.1</td>
<td>6.7</td>
<td>9.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.1</td>
<td>2.5</td>
<td>9.5</td>
<td>6.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9.5</td>
<td>2.5</td>
<td>9.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diag (i) - Diagonal Components (REAL, i=1~N)

Index(i) - Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)

Item (k) - Off-Diagonal Components (Corresponding Column ID)

AMat (k) - Off-Diagonal Components (Value)
• Sparse Matrices
• **Iterative Linear Solvers**
  − Preconditioning
  − Parallel Iterative Linear Solvers
  − Multigrid Method
  − Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Large-Scale Linear Equations in Scientific Applications

• Solving large-scale linear equations $Ax=b$ is the most important and \textit{expensive} part of various types of scientific computing.
  – for both linear and nonlinear applications

• Various types of methods proposed & developed.
  – for dense and sparse matrices
  – classified into \textit{direct} and \textit{iterative} methods

• Dense Matrices : Globally Coupled Problems
  – BEM, Spectral Methods, MO/MD (gas, liquid)

• Sparse Matrices : Locally Defined Problems
  – FEM, FDM, DEM, MD (solid), BEM w/FMP
Direct Method

- Gaussian Elimination/LU Factorization.
  - compute $\mathbf{A}^{-1}$ directly.
- Robust for wide range of applications.
- Good for both dense and sparse matrices
- More expensive than iterative methods (memory, CPU)
  - not scalable
Iterative Method

- Stationary Method
  - SOR, Gauss-Seidel, Jacobi
  - Generally Slow, Impractical

- Non-Stationary Method
  - With restriction/optimization conditions
  - Krylov-Subspace
  - CG: Conjugate Gradient
  - BiCGSTAB: Bi-Conjugate Gradient Stabilized
  - GMRES: Generalized Minimal Residual
Iterative Method (cont.)

- Less expensive than direct methods, especially in memory.
- Suitable for parallel and vector computing.

- Convergence strongly depends on problems, boundary conditions (condition number etc.)
- **Preconditioning is required**: Key Technology for Parallel FEM
Conjugate Gradient Method

- **Conjugate Gradient: CG**
  - Most popular “non-stationary” iterative method
- **Only for Symmetric Positive Definite (SPD) Matrices**
  - $\{x\}^T[A]\{x\}>0$ for arbitrary $\{x\}$
  - All of diagonal components, eigenvalues and leading principal minors $> 0$
  - Matrices of Galerkin-based FEM: heat conduction, Poisson, static linear elastic problems
- **Algorithm**
  - “Steepest Descent Method”
  - $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    - $x^{(i)}$: solution, $p^{(i)}$: search direction, $\alpha_i$: coefficient
  - Solution $\{x\}$ minimizes $\{x-y\}^T[A]\{x-y\}$, where $\{y\}$ is exact solution.
Procedures of Conjugate Gradient

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, ...$

$z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i = 1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

• Mat-Vec. Multiplication
• Dot Products
• DAXPY
Procedures of Conjugate Gradient

Compute \( r^{(0)} = b - [A]x^{(0)} \)

for \( i = 1, 2, \ldots \)

\[ z^{(i-1)} = r^{(i-1)} \]
\[ \rho_{i-1} = r^{(i-1)} z^{(i-1)} \]

if \( i = 1 \)

\[ p^{(1)} = z^{(0)} \]

else

\[ \beta_{i-1} = \rho_{i-1} / \rho_{i-2} \]
\[ p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)} \]
endif

\[ q^{(i)} = [A] p^{(i)} \]
\[ \alpha_i = \rho_{i-1} / p^{(i)} q^{(i)} \]
\[ x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} \]
\[ r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)} \]

check convergence \( |r| \)
end

• Mat-Vec. Multiplication
• Dot Products
• DAXPY

\( x^{(i)} : \text{Vector} \)
\( \alpha_i : \text{Scalar} \)
Procedures of Conjugate Gradient

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, ...$

$z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)}z^{(i-1)}$

if $i = 1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \frac{\rho_{i-1}}{p^{(i)}q^{(i)}}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)} : Vector$

$\alpha_i : Scalar$
Procedures of Conjugate Gradient

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \ldots$

$z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i = 1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

• Mat-Vec. Multiplication
• Dot Products
• DAXPY
  • Double
  • $\{y\} = a\{x\} + \{y\}$

$x^{(i)} : \text{Vector}$

$\alpha_i : \text{Scalar}$
Features of CG Method

\[
(p^{(k+1)}, Ap^{(k)}) = (r^{(k+1)} + \beta_k p^{(k)}, Ap^{(k)}) = (r^{(k+1)}, Ap^{(k)}) + \beta_k (p^{(k)}, Ap^{(k)}) = 0
\]

\[\Rightarrow \beta_k = \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})}\]

\[
(p^{(k+1)}, Ap^{(k)}) = 0 \quad p^{(k)} \text{ is “conjugate” for matrix } A
\]

Following “conjugate” relationship is obtained for arbitrary \((i,j)\):

\[
(p^{(i)}, Ap^{(j)}) = 0 \quad (i \neq j)
\]

Following relationships are also obtained for \(p^{(k)}\) and \(r^{(k)}\):

\[
(r^{(i)}, r^{(j)}) = 0 \quad (i \neq j), \quad (p^{(k)}, r^{(k)}) = (r^{(k)}, r^{(k)}), \quad r^{(k)} = b - A x^{(k)}
\]

In N-dimensional space, only N sets of orthogonal and linearly independent residual vector \(r^{(k)}\). This means CG method converges after N iterations if number of unknowns is N. Actually, round-off error sometimes affects convergence.
• Sparse Matrices
• **Iterative Linear Solvers**
  – Preconditioning
  – Parallel Iterative Linear Solvers
  – Multigrid Method
  – Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Preconditioning for Iterative Solvers

- Convergence rate of iterative solvers strongly depends on the spectral properties (eigenvalue distribution) of the coefficient matrix $A$.
  - It is preferable that distribution is small, and eigenvalues are (close to) 1 -> identity matrix

- A preconditioner $M$ transforms the linear system into one with more favorable spectral properties
  - In "ill-conditioned" problems, "condition number" (ratio of max/min eigenvalue if $A$ is symmetric) is large.
  - $M$ transforms original equation $Ax=b$ into $A'x=b'$ where $A'=M^{-1}A$, $b'=M^{-1}b$
  - Therefore if $M^{-1} \sim A^{-1}$, this is the best preconditioner (a.k.a. Gaussian Elimination)
Preconditioned Conjugate Gradient Method (PCG)

Compute \( r^{(0)} = b - [A]x^{(0)} \)
for \( i = 1, 2, \ldots \)

solve \( [M]z^{(i-1)} = r^{(i-1)} \)
\( \rho_{i-1} = r^{(i-1)} z^{(i-1)} \)
if \( i = 1 \)
\( p^{(1)} = z^{(0)} \)
else
\( \beta_{i-1} = \rho_{i-1} / \rho_{i-2} \)
\( p^{(i)} = p^{(i-1)} + \beta_{i-1} z^{(i-1)} \)
endif
\( q^{(i)} = [A]p^{(i)} \)
\( \alpha_i = \rho_{i-1} / p^{(i)} q^{(i)} \)
\( x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} \)
\( r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)} \)
check convergence \( |r| \)
end

Solving the following equation:
\[
\{z\} = [M]^{-1} \{r\}
\]

“Approximate Inverse Matrix”
\[
[M]^{-1} \approx [A]^{-1}, \quad [M] \approx [A]
\]

Ultimate Preconditioning:
Inverse Matrix
\[
[M]^{-1} = [A]^{-1}, \quad [M] = [A]
\]

Diagonal Scaling: Simple but weak
\[
[M]^{-1} = [D]^{-1}, \quad [M] = [D]
\]
Diagonal Scaling, Point-Jacobi

\[ [M] = \begin{bmatrix}
D_1 & 0 & \ldots & 0 & 0 \\
0 & D_2 & & 0 & 0 \\
& & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & D_{N-1} & 0 \\
0 & 0 & \ldots & 0 & D_N \\
\end{bmatrix} \]

- **solve** \([M]z^{(i-1)} = r^{(i-1)}\) is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c
ILU(0), IC(0)

• Widely used Preconditioners for Sparse Matrices
  – Incomplete LU Factorization
  – Incomplete Cholesky Factorization (for Symmetric Matrices)

• Incomplete Direct Method
  – Even if original matrix is sparse, inverse matrix is not necessarily sparse.
  – fill-in
  – ILU(0)/IC(0) without fill-in have same non-zero pattern with the original (sparse) matrices
LU Factorization/Decomposition: Complete LU Factorization

- A kind of direct method for solving linear eqn’s
  - compute “inverse matrix” directly
  - Information of “inverse matrix” can be saved, therefore it’s efficient for multiple RHS cases
  - “Fill-in” may occur during factorization/decomposition
    - entries which change from an initial zero to a non-zero value during the execution of factorization/decomposition

- LU factorization
Incomplete LU Factorization

- **ILU factorization**
  - Incomplete LU factorization

- **Preconditioning method using “incomplete” inverse matrices, where generation of “fill-in” is controlled**
  - Approximate/Incomplete Inverse Matrix, Weak Direct method
  - ILU(0): NO fill-in is allowed
Solving Linear Equations by LU Factorization

LU factorization of matrix $A \ (n \times n)$:

$$
\begin{pmatrix}
 a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
 a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\
 a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn}
\end{pmatrix}
= 
\begin{pmatrix}
 1 & 0 & 0 & \cdots & 0 \\
 l_{21} & 1 & 0 & \cdots & 0 \\
 l_{31} & l_{32} & 1 & \cdots & 0 \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 l_{n1} & l_{n2} & l_{n3} & \cdots & 1
\end{pmatrix}
\begin{pmatrix}
 u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
 0 & u_{22} & u_{23} & \cdots & u_{2n} \\
 0 & 0 & u_{33} & \cdots & u_{3n} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 0 & 0 & 0 & \cdots & u_{nn}
\end{pmatrix}
$$

$A = LU$

L: Lower triangular part of matrix $A$
U: Upper triangular part of matrix $A$
Example

\[ A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 8 & 7 \\ 2 & -1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & -2 \end{pmatrix} = \begin{pmatrix} 0 & 4 & 7 & 1 \\ -2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \]

\[ \Rightarrow U \]

\[ \Rightarrow L \]
Solving $Ax=b$ by LU Factorization

1. $A = LU$ \hspace{1cm} LU factorization of $A$

2. $Ly = b$ \hspace{1cm} Compute $\{y\}$ (easy) \hspace{1cm} Forward Substitution

3. $Ux = y$ \hspace{1cm} Compute $\{x\}$ (easy) \hspace{1cm} Backward Substitution

This $\{x\}$ satisfies $Ax = b$

$\therefore Ax = LUx = Ly = b$
Forward Substitution: Solving $Ly = b$

$$Ly = b$$

$$
\begin{pmatrix}
1 & 0 & \cdots & 0 \\
l_{21} & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
l_{n1} & l_{n2} & \cdots & 1 \\
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n \\
\end{pmatrix} =
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n \\
\end{pmatrix}
$$

$y_1 = b_1$

$l_{21} y_1 + y_2 = b_2$

$\vdots$

$l_{n1} y_1 + l_{n2} y_2 + \ldots + y_n = b_n$

$y_1 = b_1$

$y_2 = b_2 - l_{21} y_1$

$\vdots$

$y_n = b_n - l_{n1} y_1 - l_{n2} y_2 = b_n - \sum_{i=1}^{n-1} l_{ni} y_i$

row-by-row substitution
Backward Substitution: Solving $Ux = y$

$$Ux = y \iff \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

\begin{align*}
u_{nn} x_n &= y_n \\
u_{n-1,n-1} x_{n-1} + u_{n-1,n} x_n &= y_{n-1} \\
\vdots
\end{align*}

\begin{align*}
x_n &= \frac{y_n}{u_{nn}} \\
x_{n-1} &= \frac{(y_{n-1} - u_{n-1,n} x_n)}{u_{n-1,n-1}} \\
\vdots
\end{align*}

\begin{align*}
x_1 &= \left( y_1 - \sum_{i=2}^{n} u_{1i} x_i \right) / u_{11}
\end{align*}

row-by-row substitution
Computation of LU Factorization

\[\begin{pmatrix}
a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\
a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn}
\end{pmatrix} =
\begin{pmatrix}
1 & 0 & 0 & \cdots & 0 \\
l_{21} & 1 & 0 & \cdots & 0 \\
l_{31} & l_{32} & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{n1} & l_{n2} & l_{n3} & \cdots & 1
\end{pmatrix}\begin{pmatrix}
u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
u_{22} & u_{23} & \cdots & u_{2n} \\
u_{33} & \cdots & u_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
u_{nn}
\end{pmatrix}\]

1. \[a_{11} = u_{11}, a_{12} = u_{12}, \ldots, a_{1n} = u_{1n} \Rightarrow u_{11}, u_{12}, \ldots, u_{1n}\]
2. \[a_{21} = l_{21} u_{11}, a_{31} = l_{31} u_{11}, \ldots, a_{n1} = l_{n1} u_{11} \Rightarrow l_{21}, l_{31}, \ldots, l_{n1}\]
3. \[a_{22} = l_{21} u_{12} + u_{22}, \ldots, a_{2n} = l_{21} u_{1n} + u_{2n} \Rightarrow u_{22}, u_{23}, \ldots, u_{2n}\]
4. \[a_{32} = l_{31} u_{12} + l_{32} u_{22}, \ldots \Rightarrow l_{32}, l_{42}, \ldots, l_{n2}\]
Example: 5-Point Stencil (FDM)
Example: 5-Point Stencil (FDM)
Coef. Matrix: Diag. Component=6.00

\[
\begin{pmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 \\
\end{pmatrix} \times \begin{pmatrix}
\end{pmatrix} = \begin{pmatrix}
0.00 \\
3.00 \\
10.00 \\
11.00 \\
10.00 \\
19.00 \\
20.00 \\
16.00 \\
28.00 \\
42.00 \\
36.00 \\
52.00 \\
\end{pmatrix}
\]
# Solution

\[
\begin{pmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 6.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1.00 \\
2.00 \\
3.00 \\
4.00 \\
5.00 \\
6.00 \\
7.00 \\
8.00 \\
9.00 \\
10.00 \\
11.00 \\
12.00 \\
\end{pmatrix}
\]

= 

\[
\begin{pmatrix}
0.00 \\
3.00 \\
10.00 \\
11.00 \\
19.00 \\
20.00 \\
16.00 \\
28.00 \\
42.00 \\
36.00 \\
52.00 \\
\end{pmatrix}
\]
Complete LU Factorization

Original Matrix

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>-1.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>-1.00</td>
<td>6.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.00</td>
<td>-1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.00</td>
<td>-1.00</td>
</tr>
</tbody>
</table>

LU Factorization

Both of [L] and [U] are shown
Diag. of [L] are “1” (not shown)
fill-in occurs: some of zero components became non-zero.


### Incomp. LU fact. with no fill-in’s

Incomplete LU Factorization without fill-in’s

Both of \([L]\) and \([U]\) are shown

Diag. of \([L]\) are “1” (not shown)

---

### LU Factorization

Both of \([L]\) and \([U]\) are shown

Diag. of \([L]\) are “1” (not shown)

fill-in occurs: some of zero components became non-zero.

---

<table>
<thead>
<tr>
<th></th>
<th>6.00</th>
<th>-1.00</th>
<th>0.00</th>
<th>-1.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.17</td>
<td>5.83</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>-0.17</td>
<td>5.83</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>-0.17</td>
<td>0.00</td>
<td>0.00</td>
<td>5.83</td>
<td>-1.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>-0.17</td>
<td>0.00</td>
<td>-0.17</td>
<td>5.66</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>-0.17</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.17</td>
<td>0.00</td>
<td>-0.17</td>
<td>5.65</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>-1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>5.65</td>
<td>-1.00</td>
<td></td>
</tr>
</tbody>
</table>

---

### Incomplete LU

Incomplete LU Factorization without fill-in’s

Both of \([L]\) and \([U]\) are shown

Diag. of \([L]\) are “1” (not shown)
Slightly "Inaccurate" Solution

<table>
<thead>
<tr>
<th>6.00</th>
<th>-1.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.17</td>
<td>5.83</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>-0.17</td>
<td>5.83</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>-0.18</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Incomplete LU

Complete LU
• Sparse Matrices
• **Iterative Linear Solvers**
  – Preconditioning
  – **Parallel Iterative Linear Solvers**
  – Multigrid Method
  – Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Parallel Iterative Solvers

• Both of convergence (robustness) and efficiency (single/parallel) are important

• Global communications needed
  – Mat-Vec (P2P communications, MPI_Isend/Irecv/Waitall): Local Data Structure with HALO
    ➢ effect of latency
  – Dot-Products (MPI_Allreduce)
  – Preconditioning (up to algorithm)

• Remedy for Robust Parallel ILU Preconditioner
  – Additive Schwartz Domain Decomposition
  – HID (Hierarchical Interface Decomposition, based on global nested dissection) [Henon & Saad 2007], ext. HID [KN 2010]

• Parallel “Direct” Solvers (e.g. SuperLU, MUMPS etc.)
Local Data Structures for Parallel FEM/FDM using Krylov Iterative Solvers
Example: 2D FDM Mesh (5-point stencil)
Example: 2D FDM Mesh (5-point stencil)

4-regions/domains
Example: 2D FDM Mesh (5-point stencil)

4-regions/domains
Example: 2D FDM Mesh (5-point stencil)
meshes at domain boundary need info. neighboring domains
Example: 2D FDM Mesh (5-point stencil)

meshes at domain boundary need info. neighboring domains
Example: 2D FDM Mesh (5-point stencil)

comm. using “HALO (overlapped meshes)”
Red Lacquered Gate in 64 PEs
40,624 elements, 54,659 nodes

$k$-METIS
Load Balance = 1.03
edgecut = 7,563

$p$-METIS
Load Balance = 1.00
edgecut = 7,738
Generalized Comm. Table: Send

- Neighbors
  - NeibPETot, NeibPE[neib]
- Message size for each neighbor
  - export_index[neib], neib = 0, NeibPETot-1
- ID of **boundary** points
  - export_item[k], k = 0, export_index[NeibPETot]-1
- Messages to each neighbor
  - SendBuf[k], k = 0, export_index[NeibPETot]-1
SEND: MPI_Isend/Irecv/Waitall

```c
for (neib=0; neib<NeibPETot; neib++){
    for (k=export_index[neib]; k<export_index[neib+1]; k++){
        kk = export_item[k];
        SendBuf[k] = VAL[kk];
    }
}

for (neib=0; neib<NeibPETot; neib++){
    tag = 0;
    iS_e = export_index[neib];
    iE_e = export_index[neib+1];
    BUFlength_e = iE_e - iS_e
    ierr = MPI_Isend
        (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0, MPI_COMM_WORLD, &ReqSend[neib])
}

MPI_Waitall(NeibPETot, ReqSend, StatSend);
```
MPI_Isend

- Begins a non-blocking send
  - Send the contents of sending buffer (starting from `sendbuf`, number of messages: `count`) to `dest` with `tag`.
  - Contents of sending buffer cannot be modified before calling corresponding `MPI_Waitall`.

- `MPI_Isend`
  
  ```
  MPI_Isend(sendbuf, count, datatype, dest, tag, comm, request)
  ```

  - `sendbuf` choice I starting address of sending buffer
  - `count` int I number of elements in sending buffer
  - `datatype` MPI_Datatype I datatype of each sending buffer element
  - `dest` int I rank of destination
  - `tag` int I message tag
    This integer can be used by the application to distinguish messages. Communication occurs if `tag` of `MPI_Isend` and `MPI_Irecv` are matched.
    Usually tag is set to be “0” (in this class),

  - `comm` MPI_Comm I communicator
  - `request` MPI_Request O communication request array used in `MPI_Waitall`
MPI_Waitall

- **MPI_Waitall** blocks until all comm’s, associated with `request` in the array, complete. It is used for synchronizing `MPI_Isend` and `MPI_Irecv` in this class.
- At sending phase, contents of sending buffer cannot be modified before calling corresponding `MPI_Waitall`. At receiving phase, contents of receiving buffer cannot be used before calling corresponding `MPI_Waitall`.
- `MPI_Isend` and `MPI_Irecv` can be synchronized simultaneously with a single `MPI_Waitall` if it is consistent.
  - Same `request` should be used in `MPI_Isend` and `MPI_Irecv`.
- Its operation is similar to that of `MPI_Barrier` but, `MPI_Waitall` can not be replaced by `MPI_Barrier`.
  - Possible troubles using `MPI_Barrier` instead of `MPI_Waitall`: Contents of `request` and `status` are not updated properly, very slow operations etc.

**MPI_Waitall** (count, request, status)

- `count` `int` I number of processes to be synchronized
- `request` `MPI_Request` I/O comm. request used in `MPI_Waitall` (array size: count)
- `status` `MPI_Status` O array of status objects

`MPI_STATUS_SIZE`: defined in ‘mpif.h’, ‘mpi.h’
Generalized Comm. Table: Receive

- Neighbors
  - NeibPETot, NeibPE[neib]

- Message size for each neighbor
  - import_index[neib], neib = 0, NeibPETot-1

- ID of external points
  - import_item[k], k = 0, import_index[NeibPETot]-1

- Messages from each neighbor
  - RecvBuf[k], k = 0, import_index[NeibPETot]-1
**RECV: MPI_Isend/Irecv/Waitall**

```c
for (neib=0; neib<NeibPETot; neib++) {
    tag = 0;
    iS_i = import_index[neib];
    iE_i = import_index[neib+1];
    BUFlength_i = iE_i - iS_i

    ierr = MPI_Irecv
        (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE, NeibPE[neib], 0,
         MPI_COMM_WORLD, &ReqRecv[neib])
}

MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

for (neib=0; neib<NeibPETot; neib++) {
    for (k=import_index[neib]; k<import_index[neib+1]; k++) {
        kk = import_item[k];
        VAL[kk] = RecvBuf[k];
    }
}
```

*import_item (import_index[neib]:import_index[neib+1]-1) are received from neib-th neighbor*
MPI_Irecv

• Begins a non-blocking receive
  – Receiving the contents of receiving buffer (starting from recvbuf, number of messages: count) from source with tag.
  – Contents of receiving buffer cannot be used before calling corresponding MPI_Waitall.

• MPI_Irecv
  (recvbuf, count, datatype, source, tag, comm, request)
  - recvbuf  choice  I  starting address of receiving buffer
  - count    int    I  number of elements in receiving buffer
  - datatype  MPI_Datatype I  datatype of each receiving buffer element
  - source   int    I  rank of source
  - tag       int    I  message tag
    This integer can be used by the application to distinguish messages. Communication occurs if tag’s of MPI_Isend and MPI_Irecv are matched.
    Usually tag is set to be “0” (in this class),
  - comm      MPI_Comm I  communicator
  - request   MPI_Request O  communication request array used in MPI_Waitall
References: Libraries (mainly for flat MPI)

- Talk by the Next Speaker (Tony Drummond)

- Trillinos

- PETSc
  - http://www.mcs.anl.gov/petsc/

- GeoFEM
  - http://geofem.tokyo.rist.or.jp/

- ppOpen-HPC
  - http://ppopenhpc.cc.u-tokyo.ac.jp/
• Sparse Matrices
• **Iterative Linear Solvers**
  – Preconditioning
  – Parallel Iterative Linear Solvers
  – **Multigrid Method**
  – Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Around the multigrid in a single slide

- Multigrid is a scalable method for solving linear equations.
- Relaxation methods (smoother/smoothing operator in MG world) such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error.
- The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid.
- Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size (=scalable).
  - Good for large-scale computations
- Multigrid is also a good preconditioning algorithm for Krylov iterative solvers.
Rapid Convergence
(high-frequency error: short wave length)
Convergence of Gauss-Seidel & SOR

Slow Convergence
(low-frequency error: long wave length)
Around the multigrid in a single slide

- Multigrid is a scalable method for solving linear equations.
- Relaxation methods (smoother/smoothing operator in MG world) such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error.
- The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid.
- Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size (=scalable).
  - Good for large-scale computations
- Multigrid is also a good preconditioning algorithm for Krylov iterative solvers.
Multigrid is scalable

Weak Scaling: Problem Size/Core Fixed for 3D Poisson Eqn’s ($\Delta\phi = q$)

MGCG = Conjugate Gradient with Multigrid Preconditioning

![Graph showing iterations and seconds vs. DOF for ICCG and MGCG methods.](image)
Multigrid is scalable

Weak Scaling: Problem Size/Core Fixed

Comp. time of MGCG for weak scaling is constant:

=> scalable
Multigrid is a scalable method for solving linear equations. Relaxation methods such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error. The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid. This concept is explained here in the following simple 2-level method. If we have obtained the following linear system on a fine grid:

\[ A_F u_F = f \]

and \( A_C \) as the discrete form of the operator on the coarse grid, a simple coarse grid correction can be given by:

\[ u_F^{(i+1)} = u_F^{(i)} + R^T A_C^{-1} R ( f - A_F u_F^{(i)} ) \]

where \( R^T \) is the matrix representation of linear interpolation from the coarse grid to the fine grid (prolongation operator) and \( R \) is called the restriction operator. Thus, it is possible to calculate the residual on the fine grid, solve the coarse grid problem, and interpolate the coarse grid solution on the fine grid.
Procedure of Multigrid (2/3)

This process can be described as follows:

1. Relax the equations on the fine grid and obtain the result $u_F^{(i)} = S_F (A_F, f)$. This operator $S_F$ (e.g., Gauss-Seidel) is called the *smoothing operator* (or).
2. Calculate the residual term on the fine grid by $r_F = f - A_F u_F^{(i)}$.
3. **Restrict** the residual term on to the coarse grid by $r_C = R r_F$.
4. Solve the equation $A_C u_C = r_C$ on the coarse grid; the accuracy of the solution on the coarse grid affects the convergence of the entire multigrid system.
5. Interpolate (or *prolong*) the coarse grid correction on the fine grid by $D u_C^{(i)} = R^T u_C$.
6. Update the solution on the fine grid by $u_F^{(i+1)} = u_F^{(i)} + D u_C^{(i)}$. 
L^k W^k = F^k \quad \text{(Linear Equation: Fine Level)}

R^k = F^k - L^k w_1^k \\
v^k = W^k - w_1^k, \quad L^k v^k = R^k \\
R^{k-1} = I_{k-1}^k R^k \\
L^{k-1} v^{k-1} = R^{k-1} \quad \text{(Linear Equation: Coarse Level)}

v^k = I_{k-1}^k v^{k-1} \\
w_2^k = w_1^k + v^k

w_1^k : \text{Approx. Solution} \\
v^k : \text{Correction} \\
I_{k-1}^k : \text{Restriction Operator}

L^k W^k = F^k \quad \text{(Linear Equation: Fine Level)}

R^k = F^k - L^k w_1^k \\
v^k = W^k - w_1^k, \quad L^k v^k = R^k \\
R^{k-1} = I_{k-1}^k R^k \\
L^{k-1} v^{k-1} = R^{k-1} \quad \text{(Linear Equation: Coarse Level)}

v^k = I_{k-1}^k v^{k-1} \\
w_2^k = w_1^k + v^k

I_{k-1}^k : \text{Prolongation Operator} \\
w_2^k : \text{Approx. Solution by Multigrid}
Procedure of Multigrid (3/3)

- Recursive application of this algorithm for 2-level procedure to consecutive systems of coarse-grid equations gives a multigrid V-cycle. If the components of the V-cycle are defined appropriately, the result is a method that uniformly damps all frequencies of error with a computational cost that depends only linearly on the problem size.
  - In other words, multigrid algorithms are scalable.
- In the V-cycle, starting with the finest grid, all subsequent coarser grids are visited only once.
  - In the down-cycle, smoothers damp oscillatory error components at different grid scales.
  - In the up-cycle, the smooth error components remaining on each grid level are corrected using the error approximations on the coarser grids.
- Alternatively, in a W-cycle, the coarser grids are solved more rigorously in order to reduce residuals as much as possible before going back to the more expensive finer grids.
(a) V-Cycle

(b) W-Cycle
Multigrid as a Preconditioner

• Multigrid algorithms tend to be problem-specific solutions and less robust than preconditioned Krylov iterative methods such as the IC/ILU methods.

• Fortunately, it is easy to combine the best features of multigrid and Krylov iterative methods into one algorithm – multigrid-preconditioned Krylov iterative methods.

• The resulting algorithm is robust, efficient and scalable.

• Multigrid solvers and Krylov iterative solvers preconditioned by multigrid are intrinsically suitable for parallel computing.
Geometric and Algebraic Multigrid

- One of the most important issues in multigrid is the construction of the coarse grids.
- There are 2 basic multigrid approaches
  - geometric and algebraic
- In geometric multigrid, the geometry of the problem is used to define the various multigrid components.
- In contrast, algebraic multigrid methods use only the information available in the linear system of equations, such as matrix connectivity.
- Algebraic multigrid method (AMG) is suitable for applications with unstructured grids.
- Many tools for both geometric and algebraic methods on unstructured grids have been developed.
“Dark Side” of Multigrid Method

- Its performance is excellent for well-conditioned simple problems, such as homogeneous Poisson equations.
- But convergence could be worse for ill-conditioned problems.
- Extension of applicability of multigrid method is an active research area.
References

- https://computation.llnl.gov/casc/
• Sparse Matrices
• **Iterative Linear Solvers**
  – Preconditioning
  – Parallel Iterative Linear Solvers
  – Multigrid Method
  – Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Key-Issues for Appl’s/Algorithms towards Post-Peta & Exa Computing
Jack Dongarra (ORNL/U. Tennessee) at ISC 2013

- Hybrid/Heterogeneous Architecture
  - Multicore + GPU/Manycores (Intel MIC/Xeon Phi)
    - Data Movement, Hierarchy of Memory
- Communication/Synchronization Reducing Algorithms
- Mixed Precision Computation
- Auto-Tuning/Self-Adapting
- Fault Resilient Algorithms
- Reproducibility of Results
Recent Technical Issues in Parallel Iterative Solvers

- Communication overhead becomes significant
- Communication-Computation Overlap
  - Not so effective for Mat-Vec operations
- Communication Avoiding/Reducing Algorithms

- OpenMP/MPI Hybrid Parallel Programming Model
  - (Next section)
Communication overhead becomes larger as node/core number increases
Weak Scaling: MGCG on T2K Tokyo
Comm.-Comp. Overlapping

Internal Meshes

External (HALO) Meshes
Comm.-Comp. Overlapping

Mat-Vec operations
- Overlapping of computations of internal meshes, and importing external meshes.
- Then computation of international meshes on boundary’s
- Difficult for IC/ILU on Hybrid

- Internal Meshes
- External (HALO) Meshes
- Internal Meshes on Boundary’s
Communication Avoiding/Reducing Algorithms for Sparse Linear Solvers

• Krylov Iterative Method without Preconditioning
  – Demmel, Hoemmen, Mohiyuddin etc. (UC Berkeley)

• s-step method
  – Just one P2P communication for each Mat-Vec during $s$ iterations. Convergence becomes unstable for large $s$.
  – matrix powers kernel: $Ax$, $A^2x$, $A^3x$ ...
    • additional computations needed

• Communication Avoiding ILU0 (CA-ILU0) [Moufawad & Grigori, 2013]
  – First attempt to CA preconditioning
  – Nested dissection reordering for limited geometries (2D FDM)
Comm. Avoiding Krylov Iterative Methods using “Matrix Powers Kernel”

Figure 2. Example for PA1 and PA2. The dotted lines define the different blocks. Each block resides on a different processor. The example shows from the perspective of the processor holding the central block.

Avoiding Communication in Sparse Matrix Computations. James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, and Katherine Yelick. 2008 IPDPS
Required Information of Local Meshes for $s$-step CA computations (2D 5pt.)

$s=1$
(original)

$s=2$

$s=3$
• Sparse Matrices
• Iterative Linear Solvers
  − Preconditioning
  − Parallel Iterative Linear Solvers
  − Multigrid Method
  − Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Key-Issues for Appl’s/Algorithms towards Post-Peta & Exa Computing

Jack Dongarra (ORNL/U. Tennessee) at ISC 2013

• Hybrid/Heterogeneous Architecture
  – Multicore + GPU/Manycores (Intel MIC/Xeon Phi)
    • Data Movement, Hierarchy of Memory

• Communication/Synchronization Reducing Algorithms

• Mixed Precision Computation

• Auto-Tuning/Self-Adapting

• Fault Resilient Algorithms

• Reproducibility of Results
Motivation of This Study

• Large-scale 3D Groundwater Flow
  – Poisson equations
  – Heterogeneous porous media

• Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)

• Flat MPI vs. Hybrid (OpenMP+MPI)

• Expectations for Hybrid Parallel Programming Model
  – Number of MPI processes (and sub-domains) to be reduced
  – $O(10^8-10^9)$-way MPI might not scale in Exascale Systems
  – Easily extended to Heterogeneous Architectures
    • CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    • MPI+X: OpenMP, OpenACC, CUDA, OpenCL
Target Application: \textit{pGW3D-FVM}

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson’s equation
  - Randomly distributed water conductivity
    \[ \nabla \cdot (\lambda(x, y, z) \nabla \phi) = q, \phi = 0 \text{ at } z = z_{\text{max}} \]
  - Distribution of water conductivity is defined through methods in geostatistics \cite{Deutsch & Journel, 1998}
- Finite-Volume Method on Cubic Voxel Mesh
- Distribution of Water Conductivity
  - $10^{-5}$-$10^{+5}$, Condition Number $\sim 10^{+10}$
  - Average: 1.0
- Cyclic Distribution: $128^{3}$
Target Application: pGW3D-FVM

• 3D Groundwater Flow via. Heterogeneous Porous Media
  – Poisson’s equation
  – Randomly distributed water conductivity
    \[ \nabla \cdot (\lambda(x, y, z)\nabla \phi) = q, \phi = 0 \text{ at } z = z_{\text{max}} \]
  – Distribution of water conductivity is defined through methods in geostatistics [Deutsch & Journel, 1998]

• Finite-Volume Method on Cubic Voxel Mesh

• Distribution of Water Conductivity
  – $10^{-5}$-$10^{+5}$, Condition Number $\sim 10^{+10}$
  – Average: 1.0

• Cyclic Distribution: $128^3$
Motivation of This Study

• Large-scale 3D Groundwater Flow
  – Poisson equations
  – Heterogeneous porous media

• Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)

• Flat MPI vs. Hybrid (OpenMP+MPI)

• Expectations for Hybrid Parallel Programming Model
  – Number of MPI processes (and sub-domains) to be reduced
  – $O(10^8-10^9)$-way MPI might not scale in Exascale Systems
  – Easily extended to Heterogeneous Architectures
    • CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    • MPI+X: OpenMP, OpenACC, CUDA, OpenCL
Keywords

• Parallel Geometric Multigrid
• OpenMP/MPI Hybrid Parallel Programming Model
• Localized Block Jacobi Preconditioning
  – Overlapped Additive Schwartz Domain Decomposition (ASDD)
• OpenMP Parallelization with Coloring
• Coarse Grid Aggregation (CGA), Hierarchical CGA
Flat MPI vs. Hybrid

Flat-MPI: Each Core -> Independent

Hybrid: Hierarchical Structure
Fujitsu PRIMEHPC FX10 (Oakleaf-FX) at the U. Tokyo

- SPARC64 Ixfx (4,800 nodes, 76,800 cores)
- Commercial version of K computer
- Peak: 1.13 PFLOPS (1.043 PF, 26th, 41th TOP 500 in 2013 June.)
- Memory BWTH 398 TB/sec.
Multigrid

- Scalable Multi-Level Method using Multilevel Grid for Solving Linear Eqn’s
  - Computation Time ~ O(N) (N: # unknowns)
  - Good for large-scale problems
- Preconditioner for Krylov Iterative Linear Solvers
  - MGCG
Linear Solvers

• Preconditioned CG Method
  – Multigrid Preconditioning (MGCG)
  – IC(0) for Smoothing Operator (Smoother): good for ill-conditioned problems

• Parallel Geometric Multigrid Method
  – 8 fine meshes (children) form 1 coarse mesh (parent) in isotropic manner (octree)
  – V-cycle
  – Domain-Decomposition-based: Localized Block-Jacobi, Overlapped Additive Schwartz Domain Decomposition (ASDD)
  – Operations using a single core at the coarsest level (redundant)
Overlapped Additive Schwartz Domain Decomposition Method

ASDD: Localized Block-Jacobi Precond. is stabilized

Global Operation
\[ Mz = r \]

Local Operation
\[ z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2} \]

Global Nesting Correction
\[ z_{\Omega_1}^n = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1} (r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1}) \]
\[ z_{\Omega_2}^n = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1} (r_{\Omega_2} - M_{\Omega_2} z_{\Omega_2}^{n-1} - M_{\Gamma_2} z_{\Gamma_2}^{n-1}) \]
Computations on Fujitsu FX10

- **Fujitsu PRIMEHPC FX10 at U.Tokyo (Oakleaf-FX)**
  - 16 cores/node, flat/uniform access to memory
- **Up to 4,096 nodes (65,536 cores) (Large-Scale HPC Challenge)**
  - Max 17,179,869,184 unknowns
  - Flat MPI, HB 4x4, HB 8x2, HB 16x1
    - HB MxN: M-threads x N-MPI-processes on each node
- **Weak Scaling**
  - $64^3$ cells/core
- **Strong Scaling**
  - $128^3 \times 8 = 16,777,216$ unknowns, from 8 to 4,096 nodes
- **Network Topology is not specified**
  - 1D
HB  M  x  N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node
Reordering for extracting parallelism in each domain (= MPI Process)

- Krylov Iterative Solvers
  - Dot Products
  - SMVP
  - DAXPY
  - Preconditioning

- IC/ILU Factorization, Forward/Backward Substitution
  - Global Data Dependency
  - Reordering needed for parallelism ([KN 2003] on the Earth Simulator, KN@CMCIM-2002)
  - Multicoloring, RCM, CM-RCM
Parallereization of ICCG

**IC Factorization**

```fortran
do i = 1, N
    VAL = D(i)
    do k = indexL(i-1)+1, indexL(i)
        VAL = VAL - (AL(k)**2) * W(itemL(k), DD)
    enddo
    W(i, DD) = 1.d0 / VAL
endo
done
```

**Forward Substitution**

```fortran
do i = 1, N
    WVAL = W(i, Z)
    do k = indexL(i-1)+1, indexL(i)
        WVAL = WVAL - AL(k) * W(itemL(k), Z)
    enddo
    W(i, Z) = WVAL * W(i, DD)
endo
done
```
(Global) Data Dependency:
Writing/reading may occur simultaneously, hard to parallelize

IC Factorization

\[
\text{do } i = 1, N \\
\text{VAL} = D(i) \\
\text{do } k = \text{indexL}(i-1)+1, \text{indexL}(i) \\
\quad \text{VAL} = \text{VAL} - (\text{AL}(k)^{**2}) \times W(\text{itemL}(k), DD) \\
\text{enddo} \\
\text{W}(i, DD) = 1. d0/\text{VAL} \\
\text{enddo}
\]

Forward Substitution

\[
\text{do } i = 1, N \\
\text{WVAL} = W(i, Z) \\
\text{do } k = \text{indexL}(i-1)+1, \text{indexL}(i) \\
\quad \text{WVAL} = \text{WVAL} - \text{AL}(k) \times W(\text{itemL}(k), Z) \\
\text{enddo} \\
\text{W}(i, Z) = \text{WVAL} \times W(i, DD) \\
\text{enddo}
\]
OpenMP for SpMV: Straightforward NO data dependency

```c
!$omp parallel do private(ip, i, VAL, k)
    do ip = 1, PEsmpTOT
        do i = INDEX(ip-1)+1, INDEX(ip)
            VAL = D(i)*W(i, P)
            do k = indexL(i-1)+1, indexL(i)
                VAL = VAL + AL(k)*W(itemL(k), P)
            enddo
            do k = indexU(i-1)+1, indexU(i)
                VAL = VAL + AU(k)*W(itemU(k), P)
            enddo
            W(i, Q) = VAL
        enddo
    enddo
enddo
```
Ordering Methods
Elements in “same color” are independent: to be parallelized

MC (Color#=4) Multicoloring
RCM Reverse Cuthill-McCkee
CM-RCM (Color#=4) Cyclic MC + RCM
Ordering Methods

Elements in “same color” are independent: to be parallelized

MC (Color#=4) Multicoloring
RCM Reverse Cuthill-McKee
CM-RCM (Color#=4) Cyclic MC + RCM
What is new in this work?

- **Storage format of coefficient matrices**
  - CRS (Compressed Row Storage): Original
  - ELL (Ellpack-Itpack)

- Coarse Grid Aggregation (CGA)
- Hierarchical CGA: Communication Reducing CGA
ELL: Fixed Loop-length, Nice for Pre-fetching

\[
\begin{bmatrix}
1 & 3 & 0 & 0 & 0 \\
1 & 2 & 5 & 0 & 0 \\
4 & 1 & 3 & 0 & 0 \\
0 & 3 & 7 & 4 & 0 \\
1 & 0 & 0 & 0 & 5 \\
\end{bmatrix}
\]

(a) CRS  
(b) ELL
Special Treatment for “Boundary” Cells connected to “Halo”

- Distribution of Lower/Upper Non-Zero Off-Diagonal Components
  - Pure Internal Cells
    - L: ~3, U: ~3
  - Boundary Cells
    - L: ~3, U: ~6

![Diagram showing the distribution of internal, external, and boundary cells.](image)

- External Cells
- Internal Cells
- Pure Internal Cells on Boundary
Effect of CRS/ELL

4 nodes, 64 cores, (16,777,216 meshes: $64^3$ meshes/core)
CM-RCM(k), only RCM for ELL cases

**Down is good**
## Analyses by Detailed Profiler of Fujitsu FX10, single node, Flat MPI, RCM (Multigrid Part)
ELL with fixed loop length: accel. prefetching

<table>
<thead>
<tr>
<th></th>
<th>L1-cache Demand Miss</th>
<th>Instructions</th>
<th>Time for Multigrid</th>
<th>Operation Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRS</td>
<td>29.3%</td>
<td>$1.447 \times 10^{10}$</td>
<td>6.815 sec.</td>
<td>1.453 sec.</td>
</tr>
<tr>
<td>ELL</td>
<td>16.5%</td>
<td>$6.385 \times 10^{9}$</td>
<td>5.457 sec.</td>
<td>0.312 sec.</td>
</tr>
</tbody>
</table>
**Original Approach (restriction)**

Coarse grid solver at a single core [KN 2010]

<table>
<thead>
<tr>
<th>Level</th>
<th>Mesh # for each MPI</th>
<th>Coarse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level=2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level=m-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level=m-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level=m-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level=m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Coarse grid solver on a single core (further multigrid)
Coarse Grid Solver on a Single Core

Original Approach

Size of the Coarsest Grid = Number of MPI Processes
In Flat-MPI, this size is larger

lev=1
lev=2
lev=3
lev=4

PE#0
PE#1
PE#2
PE#3

ISS-2013
Original Approach (restriction)

Coarse grid solver at a single core [KN 2010]

Level=1
Level=2
Level=m-3
Level=m-2
Level=m-1
Level=m

Mesh # for each MPI= 1

Communication Overhead at Coarser Levels

Coarse grid solver on a single core (further multigrid)
Coarse Grid Aggregation (CGA)

Coarse Grid Solver is multithreaded [KN 2012]

- Communication overhead could be reduced
- Coarse grid solver is more expensive than original approach.
- If process number is larger, this effect might be significant

Coarse grid solver on a single MPI process (multithreaded, further multigrid)
Results at 4,096 nodes

*lev*: switching level to “coarse grid solver”

Opt. Level= 7, HB 8x8 is the best

DOWN is GOOD

**HB 8x2**

- ELL-CGA, lev=6: 51
- ELL-CGA, lev=7: 55
- ELL-CGA, lev=8: 60
- ELL: 65
- CRS: 66

Switching Level for Coarse Grid Solver

**HB 16x1**

- ELL-CGA, lev=6: 51
- ELL-CGA, lev=7: 55
- ELL-CGA, lev=8: 60
- ELL-CGA, lev=9: 63
- ELL: 65
- CRS: 66

Switching Level for Coarse Grid Solver

Down is good
Weak Scaling at 4,096 nodes
17,179,869,184 meshes (64³ meshes/core)
best switching level (=7)

Down is good
Weak Scaling: up to 4,096 nodes
up to 17,179,869,184 meshes (64^3 meshes/core)
Convergence has been much improved by coarse grid aggregation, **DOWN is GOOD**

**Iterations**

**sec.**

- Flat MPI: ELL
- HB 8x2: CRS
- HB 8x2: ELL
- HB 8x2: ELL-CGA

**CORE#**

**CORE#**

**Down is good**
Strong Scaling at 4,096 nodes
268,435,456 meshes, only $16^3$ meshes/core at 4,096 nodes

Down is good

Flat MPI/ELL, 8 nodes (128 cores) is 100%

<table>
<thead>
<tr>
<th></th>
<th>Flat MPI</th>
<th>HB 4×4</th>
<th>HB 8×2</th>
<th>HB 16×1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRS</td>
<td>ELL</td>
<td>CRS</td>
<td>ELL-CGA</td>
</tr>
<tr>
<td>Iterations until Convergence</td>
<td>57</td>
<td>58</td>
<td>58</td>
<td>46</td>
</tr>
<tr>
<td>MGCG solver (sec.)</td>
<td>5.73</td>
<td>4.07</td>
<td>1.38</td>
<td>.816</td>
</tr>
<tr>
<td>Parallel performance (%)</td>
<td>2.02</td>
<td>2.85</td>
<td>8.38</td>
<td>14.2</td>
</tr>
</tbody>
</table>
Hierarchical CGA: Comm. Reducing MG

Reduced number of MPI processes [KN 2013]

Fine

Level=1

Level=2

Level=m-3

Level=m-3

Level=m-2

Coarse grid solver at a single MPI process (multi-threaded, further multigrid)
Results at 4,096 nodes

lev: switching level to “coarse grid solver”

Opt. Level= 7, HB 8x8 is the best

DOWN is GOOD

HB 8x2

HB 16x1

Switching Level for Coarse Grid Solver

Down is good
Summary

• ELL format is effective!
• “Coarse Grid Aggregation (CGA)” is effective for stabilization of convergence at $O(10^4)$ cores for MGCG
  – HB 8x2 is the best at 4,096 nodes
• Hierarchical CGA ($h$CGA) is also effective at 4,096 nodes

• Future/On-Going Works and Open Problems
  – Algorithms
    • CA-Multigrid (for coarser levels), CA-SPAI
  – Strategy for Automatic Selection
    • optimum switching level, number of processes for $h$CGA, optimum color #
  – More Flexible ELL for Unstructured Grids
  – Optimized MPI (co-design)
    • e.g. MPI on Fujitsu FX10 utilizing RDMA with persistent communications
  – Optimum number of colors
    • strongly depends on thread #, H/W etc...
• Sparse Matrices
• Iterative Linear Solvers
  – Preconditioning
  – Parallel Iterative Linear Solvers
  – Multigrid Method
  – Recent Technical Issues
• Example of Parallel MGCG
• ppOpen-HPC
Current Supercomputer Systems
University of Tokyo

- Total number of users ~ 2,000
- Hitachi HA8000 Cluster System (T2K/Tokyo) (2008.6-)
  - Cluster based on AMD Quad-Core Opteron (Barcelona)
  - 140.1 TFLOPS
- Hitachi SR16000/M1 (Yayoi) (2011.10-)
  - Power 7 based SMP with 200 GB/node
  - 54.9 TFLOPS
- Fujitsu PRIMEHPC FX10 (Oakleaf-FX) (2012.04-)
  - SPARC64 IXfx
  - Commercial version of K computer
  - 1.13 PFLOPS (1.043 PFLOPS for LINPACK, 21st in 40th TOP500)
Supercomputers in U.Tokyo

FY
05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20

Hitachi SR11000/J2
18.8 TFLOPS, 16.4 TB
SMP, Automatic Parallelization

Hitachi SR16000/M1 based on IBM Power-7
54.9 TFLOPS, 11.2 TB
Our last SMP, Users have to learn MPI ...

Hitachi HA8000 (T2K)
140 TFLOPS, 31.3 TB
MPI, well-balanced system in spite of slow memory

Fujitsu PRIMEHPC FX10 based on SPARC64 IXfx
1.13 PFLOPS, 150 TB
Turning-point to Hybrid, also good MPI performance

Post T2K/Heterogeneous Computing Nodes
O(10^1-10^2)PFLOPS

HOP
STEP
JUMP

Peta
K
Exa
Post T2K System

- Will be installed FY.2014-2015, O(10^1-10^2) PFLOPS
  - under collaboration with U. Tsukuba

- Heterogeneous computing node will be adopted
  - best performance and well balanced memory-computation under limited power consumption.

- Multi-core CPU+GPU, Multi-core CPU+Many-core (e.g. Intel MIC/Xeon Phi)
  - TSUBAME 2.0 (Tokyo Tech)
  - HA-PACS (U.Tsukuba)
  - We are mainly thinking about MIC/Xeon-Phi-based system.

- Programming is difficult
  - (MPI+OpenMP) is already difficult
    - Explicit method is rather easier
      - OpenACC, CUDA, OpenCL

- OpenMP & OpenACC for MIC
ppOpen-HPC (1/3)

- Open Source Infrastructure for development and execution of large-scale scientific applications on post-peta-scale supercomputers with automatic tuning (AT)
  - “pp” : post-peta-scale

- Five-year project (FY.2011-2015) (since April 2011)
  - P.I.: Kengo Nakajima (ITC, The University of Tokyo)
  - Part of “Development of System Software Technologies for Post-Peta Scale High Performance Computing” funded by JST/CREST (Japan Science and Technology Agency, Core Research for Evolutonal Science and Technology)
  - 4.5 M$ for 5 yr.

- Team with 6 institutes, >30 people (5 PDs) from various fields: Co-Designin
  - ITC/U.Tokyo, AORI/U.Tokyo, ERI/U.Tokyo, FS/U.Tokyo
  - Kyoto U., JAMSTEC
ppOpen-HPC (2/3)

• ppOpen-HPC consists of various types of optimized libraries, which covers various types of procedures for scientific computations.
  • FEM, FDM, FVM, BEM, DEM

• Source code developed on a PC with a single processor is linked with these libraries, and generated parallel code is optimized for post-peta scale system.

• Users don’t have to worry about optimization tuning, parallelization etc.
  • CUDA, OpenGL etc. are hidden.
  • Part of MPI codes are also hidden.
  • OpenMP, OpenACC could be hidden
ppOpen-HPC covers ...
Program My_pFEM
use ppOpenFEM_util
use ppOpenFEM_solver

call ppOpenFEM_init
call ppOpenFEM_cntl
call ppOpenFEM_mesh
call ppOpenFEM_mat_init

do
    call ppOpenFEM_mat_ass
call ppOpenFEM_mat_bc
call ppOpenFEM_solve
call ppOpenFEM_vis
    Time = Time + DT
dendo
call ppOpenFEM_finalize
stop
end
User’s Program

ppOpen-APPL
ppOpen-MATH
ppOpen-AT
ppOpen-SYS
ppOpen-HPC

FEM  FDM  FVM  BEM  DEM
MG  GRAPH  VIS  MP
STATIC  DYNAMIC
COMM  FT

Optimized Application with
Optimized ppOpen-APPL, ppOpen-MATH
ppOpen-HPC (3/3)

- Capability of automatic tuning (AT) enables development of optimized codes and libraries on emerging architecture based on results by existing architectures and machine parameters.
  - Solvers & Libraries in ppOpen-HPC
  - OpenFOAM, PETSc etc.
- Target system is post-peta-scale computer with heterogeneous computing nodes which consist of multicore CPU’s and co-processors, such as GPU’s and manycores.
  - Peak performance is $O(10^1-10^2)$ PFLOPS, and number of cores are $O(>10^6)$ cores.
  - Post T2K (MIC-based) to be installed in FY.2014-2015
  - ppOpen-HPC helps smooth transition of users to new system
Schedule of Public Release
(with English Documents)
We are now focusing on MIC/Xeon Phi

• 4Q 2012
  – ppOpen-HPC for Multicore Cluster (Cray, K etc.)
  – Preliminary version of ppOpen-AT/STATIC
    • available in SC’12

• 3Q 2013
  – ppOpen-HPC for Multicore Cluster & Xeon Phi (& GPU)

• 3Q 2014
  – Prototype of ppOpen-HPC for Post-Peta Scale System

• 4Q 2015
  – Final version of ppOpen-HPC for Post-Peta Scale System
  – Further optimization on the target system
ppOpen-HPC v.0.1.0
http://ppopenhpc.cc.u-tokyo.ac.jp/

- Released at SC12 (or can be downloaded)
- Multicore cluster version (Flat MPI, OpenMP/MPI Hybrid) with documents in English

<table>
<thead>
<tr>
<th>Component</th>
<th>Archive</th>
<th>Flat MPI</th>
<th>OpenMP/MPI</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppOpen-APPL/FDM</td>
<td>ppohFDM_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppOpen-APPL/FVM</td>
<td>ppohFVM_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppOpen-APPL/FEM</td>
<td>ppohFEM_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppOpen-APPL/BEM</td>
<td>ppohBEM_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppOpen-APPL/DEM</td>
<td>ppohDEM_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppOpen-MATH/VIS</td>
<td>ppohVIS_FDM3D_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppOpen-AT/STATIC</td>
<td>ppohAT_0.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>